Definition of Formats for Coding Phylogenetic Trees in R—DRAFT

Emmanuel Paradis

June 14, 2006

Contents

1 Introduction

This document explains how phylogenetic trees are coded and handled in the R package `ape`. Such data, like any R data, can be passed to C code for computing-intensive tasks. How these data can be manipulated efficiently in R, and how they can be extended is discussed.

Phylogenetic trees are complex data structures: coding them in computer programs requires special care. Felsenstein [2] outlines the coding used in most computer programs for phylogenetic analysis. Such data structures cannot be used easily in an interactive way because they are based on the recursive use of pointers. In addition, they cannot be extended or modified without recompiling the whole program. In R a different approach is used based on simple data structures like matrices and vectors. The class `"hclust"`, used by the function

`hclust`, codes hierarchical clusters with a two-column matrix indicating the pairings of observations. To code phylogenetic trees in R, it was necessary to extend and modify this class significantly.

The class `"phylo"` introduced in `ape` in August 2002 has this aim. Its initial definition in version 0.1 of `ape` has proven of some utilities in some applications. These applications and some feed-back from users have pointed out to some weaknesses and limitations of this initial definition.

In this document, I detail the new definition of the class `"phylo"` introduced in `ape` 1.8-4. From this new definition, I discuss several issues which can be sketched as follows.

1. Manipulating efficiently the tree and associated data in R and C.
2. Relating nodes and tips to other data (vectors, data frames, ...).
3. Tracking a node when tips are deleted from or added to a tree.
4. Uniqueness of representation.

2 Terminology and Notations

branch:	edge, vertex
node:	internal node
tip:	terminal node
n:	number of tips
m:	number of nodes

3 Definition of the Class `"phylo"`

An object of class `"phylo"` is a list with, at least, the following *mandatory* elements:

1. A numeric matrix named `edge` with two columns and as many rows as there are branches in the tree;
2. An attribute `class` equal to `"phylo"`.

In the matrix `edge`, each branch is coded by the nodes it connects: internal nodes are coded with negative numbers ($-1, \ldots, -m$; -1 is the root), and tips are coded with positive numbers ($1, \ldots, n$). Both series are numbered with no gaps.

The matrix `edge` has the following properties:

- The first column has only negative values (thus, positive values appear only in the second column).
- All nodes appear in the first column at least twice.
- The number of occurrences of a node in the first column is related to the nature of the node: twice if it is dichotomous, three times if it is trichotomous, and so on.

- All elements, except the root -1, appear once in the second column (only if the tree has no reticulation).

This representation is applied to rooted and unrooted trees. For the latter, the position of the root is arbitrary.

The smallest tree of class `"phylo"` can be created in R with:

Q: do we permit trees with only one branch, `edge <- matrix(c(-1, 1), 1)`?

```
> tr <- list(edge = matrix(c(-1, -1, 1, 2), 2, 2))
> class(tr) <- "phylo"
> str(tr)

List of 1
 $ edge: num [1:2, 1:2] -1 -1 1 2
 - attr(*, "class")= chr "phylo"
```

The following elements are *optional*:

1. A numeric vector named `edge.length` with the branch lengths: this has as many values of the number of rows of `edge`;
2. A character vector of length n named `tip.label` with the labels of the tips;
3. A character vector of length m named `node.label` with the labels of the nodes;
4. A single numeric value named `root.edge` giving the length of the branch at the root.

For the three first of these, there is a correspondence between their elements and the structure of `edge` (e.g., the length of the ith branch `edge[i, ]` is given by `edge.length[i]`).

There are two substantial changes compared to the definition used until version 1.8-3 of `ape`: (i) `edge` was previously of mode character, and (ii) `tip.label` was previously mandatory. Compatibily is thus straightforward, for instance to convert our tree `tr` to the old definition:

```
> mode(tr$edge) <- "character"
> if (is.null(tr$tip.label)) tr$tip.label <- rep("", 2)
> str(tr)

List of 2
 $ edge     : chr [1:2, 1:2] "-1" "-1" "1" "2"
 $ tip.label: chr [1:2] "" ""
 - attr(*, "class")= chr "phylo"
```

Whereas conversion from the old definition to the new one is easier:

```
> mode(tr$edge) <- "numeric"
> str(tr)

List of 2
 $ edge     : num [1:2, 1:2] -1 -1 1 2
 $ tip.label: chr [1:2] "" ""
 - attr(*, "class")= chr "phylo"
```

Note that this new definition still uses the S3 classes, though a switch to the S4 classes may be envisaged in the future.

3.1 Memory Requirements

The table below gives the sizes as given by `object.size()` of three phylogenies distributed with `ape` and often used in examples, and two random trees generated with `rtree`. The column labelled "old" gives the size in bytes with the old definition of the class `"phylo"`, and the one labelled "new" gives the size using the new definition.

tree	n	old	new	gain
bird.orders	23	5560	2744	2.02
bird.families	135	30,992	13,648	2.27
chiroptera	917	158,900	72,884	2.18
random tree	1000	216,396	88,524	2.44
"	10,000	2,160,396	880,524	2.45

The gain in terms of memory requirements is thus at least twice. Less than 1 Mb is needed to store a tree of 10,000 tips with the new definition. Given than most computers have nowadays at least 1 Gb of RAM, this should make a wide range of analyses feasible.

3.2 Guidelines for Creating the Matrix `edge`

In the matrix `edge`, each row represents a branch: the node in the first column is the origin of the branch, and the node or tip in the second column is its end. Note that for unrooted trees, this order is arbitrary (except for the terminal branches) because the position of the root is also arbitrary. This representation allows a lot of generalizations, such as multichotomies ...

```
> matrix(c(rep(-1, 3), 1:3), 3, 2)

     [,1] [,2]
[1,]   -1    1
[2,]   -1    2
[3,]   -1    3
```

... or reticulations:

```
> a <- c(-1, -2, -2, -2, -1, -3, -3)
> b <- c(-2, 1, 2, -3, -3, 3, 4)
> cbind(a, b)

      a  b
[1,] -1 -2
[2,] -2  1
[3,] -2  2
[4,] -2 -3
[5,] -1 -3
[6,] -3  3
[7,] -3  4
```

Q: These reticulations may be stored in a distinct matrix to avoid confusion.

There is an arbitrary direction ($-2 \rightarrow -3$), from the largest number (or smallest in absolute value) to the smallest one.

There is no mandatory order for the rows of `edge`, but they may be arranged in a way that is efficient for computation and manipulation. For instance, consider the tree in Newick format:

```
"((,),(,));"
```

Then the two following matrices are similar for `edge`:

```
> cbind(c(-1, -2, -2, -1, -3, -3), c(-2, 1, 2, -3, 3, 4))
```

```
     [,1] [,2]
[1,]   -1   -2
[2,]   -2    1
[3,]   -2    2
[4,]   -1   -3
[5,]   -3    3
[6,]   -3    4
```

```
> cbind(c(-1, -1, -2, -2, -3, -3), c(-2, -3, 1, 2, 3, 4))
```

```
     [,1] [,2]
[1,]   -1   -2
[2,]   -1   -3
[3,]   -2    1
[4,]   -2    2
[5,]   -3    3
[6,]   -3    4
```

In the first representation the branches are grouped *clade-wise*, whereas in the second one the internal branches come first. Any order of the rows are valid with respect to the above definition. However, the clade-wise order has an interesting feature: it is straightforward to find all the branches descendant of a given node (see § 4.2).

There is another interesting order:

```
> cbind(c(-2, -2, -3, -3, -1, -1), c(1, 2, 3, 4, -2, -3))
```

```
     [,1] [,2]
[1,]   -2    1
[2,]   -2    2
[3,]   -3    3
[4,]   -3    4
[5,]   -1   -2
[6,]   -1   -3
```

Here, the branches are arranged so that a "pruning" calculation (or postorder tree traversal) can be done by reading down the rows of `edge`. Additionally, if some conventions are taken, this arrangement can lead to a unique representation for a given tree in the same way than the matchings proposed by Diaconis & Holmes [1].[1] I shall call the above order *pruning-wise*.

[1] Matchings work only for rooted dichotomous trees.

4 Tree Manipulation

The table below shows how to perform a few basic operations on objects of class `"phylo"` in R.

How many tips?	`max(tr$edge)`
How many nodes?	`-min(tr$edge)`
How many branches?	`dim(tr$edge)[1]`
How to find node `x`?	`which(tr$edge == x)` *or* `which(tr$edge == x, TRUE)`
What is the ancestor of node `x`?	`i <- which(tr$edge[, 2] == x)` `tr$edge[i, 1]`
What are the terminal branches?	`which(tr$edge[, 2] > 0)`

Of course, each of these commands may be wrapped in a function, for instance:

```
getNtips <- function(phy) max(phy$edge)
```

4.1 Preorder Tree Traversal

Preorder tree traversal means here: travelling through the tree from the root to the tips. If an object of class `"phylo"` is in clade-wise order, then the first element of the first column of `edge` is, by definition, the root and numbered -1. This is true whether the tree is rooted or not (remind that the root is arbitrary in the latter case). The beginning and end of each clade connected to the root is found with:

```
start <- which(tr$edge[, 1] == -1)
end <- c(start[-1] - 1, dim(tr$edge)[1])
```

The MRCA of these clades (i.e., the direct descendants of the root) can be found with:

```
tr$edge[start, 2]
```

As an example, we take the avian families tree:

```
> start <- which(bird.families$edge[, 1] == -1)
> end <- c(start[-1] - 1, dim(bird.families$edge)[1])
> start

[1]  1 28

> end

[1]  27 271
```

The two nodes connected to the root are:

```
> bird.families$edge[start, 2]

[1]  -2 -15
```

This can be checked graphically with:

```
plot(bird.families)
nodelabels()
```

This approach can then be applied repeatedly from the root to the tips. For instance, we find that the first node descendant of the root is `tr$edge[start[1], 2]`: we may thus replace "`-1`" above by this value, and "`dim(tr$edge)[1]`" by "`end[1]`":

```
startB <- which(tr$edge[, 1] == tr$edge[start[1], 2])
endB <- c(startB[-1] - 1, end[1])
```

Note the new names `startB` and `endB`; in practice, this may be simplified by using, for instance, recursive calls to a function (see § 4.8).

If the object `tr` is in pruning-wise order, then tree traversal may be done through successive search of node and tips numbers using `which` (as was done by most functions in `ape`). However, this is less efficient.

4.2 Finding a Clade

For an arbitrary node, say `nod`, the approach above may be adapted to the following algorithm:

1. Find the node ancestor of `nod`, store its number in `anc`, and store the number of the branch in `i`.
2. Find the next occurrence of `anc` in `tr$edge[, 1]`, store it in `j`.

The clade descending from `nod` is given by the rows $i+1$ to $j-1$ of `tr$edge`. This algorithm in R is:

```
i <- which(tr$edge[, 2] == nod)
anc <- tr$edge[i, 1]
tmp <- which(tr$edge[, 1] == anc)
j <- tmp[which(tmp == i) + 1]
tr$edge[(i+1):(j-1), ]
```

Note that it is straightforward to translate this code in C.

4.3 Postorder Tree Traversal

Postorder tree traversal means here: travelling through the tree from the tips to the root. If the object `tr` is in pruning-wise order, then postorder tree traversal is straightforward by descending along the rows of `tr$edge`. Otherwise, successive searches must be done.

4.4 Passing Trees from R to C

Because the class `"phylo"` uses only vectors,[2] it is easy to pass these data to C using the R function `.C`. For instance, the matrix `edge` may be passed with:

[2] Matrices in R are actually vectors.

```
.C("nameofCfunction", as.integer(tr$edge),
   as.integer(dim(tr$edge)[1]), PACKAGE = "nameofpackage")
```

which will be received in C with:

```
void nameofCfunction(int * edge, int * n)
```

where `edge` is a pointer to an array of size $2n$. The two nodes of the ith branch will be accessed with `edge[i - 1]` and `edge[i - 1 + n]`.

An alternative, maybe easier, solution is to pass separately the two columns of `edge`:

```
.C("nameofCfunction", as.integer(tr$edge[, 1]),
   as.integer(tr$edge[, 2]),
   as.integer(dim(tr$edge)[1]), PACKAGE = "nameofpackage")
```

with in the C program:

```
void nameofCfunction(int * edge1, int * edge2, int * n)
```

Now the two nodes will be accessed with `edge1[i - 1]` and `edge2[i - 1]`.

A complete tree structure may be passed with `.C`:

```
.C("nameofCfunction", as.integer(tr$edge),
   as.integer(dim(tr$edge)[1]), as.double(tr$edge.length),
   as.character(tr$tip.label), as.character(tr$node.label),
   PACKAGE = "nameofpackage")
```

received in C with:

```
void nameofCfunction(int * edge, int * n, double * edge_length,
                     char ** tip_label,  char ** node_label)
```

Note that other elements may be added in the arguments as long as they are R vectors. The advantage of using `.C` is that data manipulation in C is similar to any program in this language: the only constraint is that the function receiving the data from R must have pointers and return void.

Using `.Call` or `.External` is more flexible on the R side:

```
.Call("nameofCfunction", tr, PACKAGE = "nameofpackage")
.External("nameofCfunction", tr, PACKAGE = "nameofpackage")
```

where `tr` may be any R data. But the data manipulation in C is more complex since it deals with SEXP (*S expression*) structures:

```
SEXP nameofCfunction(SEXP tr)
```

This is advantegeous if the number and/or size of elements is unknown in advance. An example can be found in the sources of `ape` (see src/bipartition.c and R/dist.topo.R).

4.5 Relating Nodes and Tips to Other Data

Use numeric indexing (more efficient) if possible, otherwise use `names` or any other way to assign the value to its node or tip.

4.6 Tracking Nodes

The problem of tracking a node through successive tree manipulation is not easy because nodes are numbered sequentially. For instance, if two trees are binded, the node numbers of one of them must be changed. The solution to this problem is to use the element `node.label`. Node labels may be created easily with:

```
tr$node.label <- paste("node", 1:(-min(tr$edge)), sep = "")
```

If a second tree, say `trb`, is involved here:

```
trb$node.label <- paste("trb_node", 1:(-min(tr$edge)), sep = "")
```

Both trees may be binded now.

```
trc <- bind.tree(tr, trb)
```

To find the node number of say `"trb_node1"` in the new tree:

```
which(trc$node.label == trb_node1)
```

4.7 Uniqueness of Representation

At the moment, this remains unsolved since even adopting one of the rules described above for the order of the branches may lead to several representations of the same tree. Additional rules are needed. Some open questions:

- Is there an order that makes tree manipulation optimal? (see above for some partial answers)
- Do we really need a standard, unique order?

Keep in mind that the rules should work for rooted and unrooted, dichotomous and multichotomous trees, as well as reticulograms.

4.8 Example

As an example of using the new definition of the class `"phylo"` and the guidelines above, I rewrote the function `rtree`. The logic of the algorithm is the same in both versions: splitting repeatedly a pool of n tips until $n = 1$. But the implementations in R are different. In the old version, the branches were defined successively and the matrix `edge` was filled along its rows; the latter were reordered at the end. In the new version, the rows are filled with respect to the size of the clades using a recursive function, so that no reordering is needed at the end. The following table compares the timings of the two versions.[3]

n	old	new	gain
10	0.026	0.0006	43
100	0.290	0.003	100
1000	4.017	0.030	134

[3] Timings averaged over 100 consecutive repetitions with a `for` loop, processor at 1.86 GHz, 2 Gb RAM, Linux Knoppix 4.0, R 2.3.0, `ape` 1.8-3.

This shows that considerable improvement can be achieved even using R code only. Furthermore, the algorithm is now more appropriate to be coded in C, so that more improvement seems possible. Note that the operation done by `rtree` is actually a preorder tree traversal: thus, this operation can be done in a very short time with R code.

5 Others

It is also possible to add further attributes (optionally?) to an object of class `"phylo"` such as the number of tips (n) or the number of nodes (m), ... also `rooted` (TRUE or FALSE), `order` (`"cladewise"` or `"pruningwise"`), and so on. This may use `attr` or `$`.

In fact, any element may be appended in an object of class `"phylo"` since it is a list. Adding some information may be useful to store the results of previous computations (as done with the class `"phylog"` in `ade4`).

6 References

[1] Diaconis P. W. & Holmes S. P. 1998. Matchings and phylogenetic trees. *Proceedings of the National Academy of Sciences USA* **95**: 14600–14602.

[2] Felsenstein J. 2004. *Inferring phylogenies*. Sinauer Associates, Sunderland, Mass., USA.